Social-Media-Plattform

Aufgaben

Eine Social-Media-Plattform ermöglicht ihren Nutzerinnen und Nutzern (im folgenden Nutzer genannt), sich digital zu vernetzen und sich untereinander auszutauschen. Nutzer können selbst Beiträge erstellen und die Beiträge anderer Nutzer einsehen. Ein Nutzer kann andere Nutzer abonnieren. Auf deren neue Beiträge wird er nach seiner Anmeldung am System besonders hingewiesen. Jeder Nutzer kennt seine Abonnenten.
Für eine Social-Media-Plattform sollen ein Client-Server-System und eine Datenbank entwickelt werden.

1 Objektorientierte Entwicklung eines Softwaresystems für die Social-Media-Plattform
Ein erstes UML-Klassendiagramm ist in Material 1 zu finden.

1.1 Zwischen der Klasse `Beitrag` und den Klassen `Bild` und `Text` (Material 1) sind "besteht aus"-Assoziationen modelliert. Benennen Sie die beiden Arten von Assoziationen und erläutern Sie jeweils deren Bedeutung.

(3 BE)

1.2 Überführen Sie die Klassen `Nutzer` und `Beitrag` in Anweisungen einer objektorientierten Programmiersprache und implementieren Sie die Konstruktoren und Methoden.

Hinweise: Alle Aktionen des Nutzers führen zu einer Aktualisierung seiner letzten Aktivität. Ein Nutzer kann seine eigenen Beiträge nicht liken. Abonniert ein Nutzer einen anderen Nutzer, soll er nur einmal als Abonnent bzw. als abonnierter Nutzer gespeichert werden. Ein Benutzer kann sich nicht selbst abonnieren. Beim Abonnieren eines Nutzers wird die bidirektionale Abonnentenbeziehung vollständig implementiert. Eine Beschreibung der Klasse `List` ist in Material 2 zu finden.

(13 BE)

1.3 Die Methode `ermittleAbonnierteNutzerMitNeuenBeitraegen(n: Nutzer)` der Klasse `SocialMediaPlatform` (Material 1) ermittelt alle Nutzer, die der Nutzer `n` abonniert hat und die neue Beiträge erstellt haben, seit er zuletzt aktiv war. Entwickeln und zeichnen Sie ein Struktogramm für diese Methode.

Hinweis: Ein abonnierter Nutzer soll nur einmal in der Liste enthalten sein, auch wenn er mehrere Beiträge im relevanten Zeitraum erstellt hat.

(7 BE)

1.4 Das Softwaresystem wird als Client-Server-System realisiert.
Nachdem Client und Server eine Verbindung aufgebaut haben, muss sich ein Nutzer auf dem Client mit Namen und Passwort anmelden, der Client sendet das Kommando "anmelden" an den Server. Nach erfolgreicher Anmeldung sendet der Server eine Willkommens-Nachricht.
Der Beginn einer Sitzung ist im Folgenden wiedergegeben:

```
<anmelden;Anna;%fivqwj2iKez<LF>
>+OK Willkommen <LF>
```

Hinweise: Das Zeichen < kennzeichnet die Anforderung vom Client, > kennzeichnet die Antwort vom Server, <LF> steht für Line Feed (`'\n'`). Anna ist der Nutzername, %fivqwj2iKez das Passwort. Die Socket-Klassen sind in Material 2 beschrieben.

1.4.1 Beschreiben Sie die Aufgaben und Funktionsweisen der Socket-Klassen, der Klassen `Server` und `ServerThread` sowie deren Zusammenarbeit mit der Klasse `SocialMediaPlatform` beim Verbindungsaufbau und der Anmeldung.

(4 BE)

1.4.2 Entwickeln und zeichnen Sie ein Sequenzdiagramm in UML-Notation unter Verwendung der Vorlagen in Material 3, das den Verbindungsaufbau und den dargestellten Ablauf bis zu Annas erfolgreicher Anmeldung darstellt.

Hinweise: Die beiden Seiten der Sequenzdiagrammvorlagen sind aneinander zu legen. Es ist davon auszugehen, dass ein Objekt `n` der Klasse `Nutzer` bereits existiert. Eine Fehlerbehandlung ist nicht darzustellen.

(10 BE)

1.4.3 Überführen Sie die Klasse `Server` (Material 1) in eine objektorientierte Programmiersprache.

(5 BE)

1.5 Wenn ein Nutzer sein Passwort vergessen hat, kann er ein neues anfordern. Die Methode `generierePasswort()` der Klasse `SocialMediaPlatform` liefert ein sicheres Passwort, das zwölf Zeichen lang ist und die folgenden Elemente enthält:
– einen Großbuchstaben [A – Z]
– eine Ziffer [0 – 9]
– ein Sonderzeichen [#, $, %, &]
Jedes Element wird per Zufall ausgewählt und an eine, ebenfalls per Zufall ausgewählte Position im 12-stelligen Passwort geschrieben. Die restlichen neun Stellen werden mit zufälligen Kleinbuchstaben besetzt.
Beispiele für Passwörter sind `%fivqwj2iKez`, `i4koarmbN$ae`, `Sbazoygvc4#s`.

Hinweis: In Material 4 ist eine ASCII-Tabelle zu finden, in Material 2 die Beschreibung der Klasse `Random`, mit der Zufallszahlen erzeugt werden können.

1.5.1 Entwickeln Sie einen Algorithmus für die Methode `generierePasswort()` in Textform oder als Pseudocode.

(5 BE)

1.5.2 Implementieren Sie die Methode `generierePasswort()` nach Ihrem Algorithmus aus Aufgabe 1.5.1.

(8 BE)

1.6 Die Methode `registrieren()` der Klasse `SocialMediaPlattform` erzeugt einen neuen Nutzer und trägt diesen in die Liste der Nutzer ein, sofern der Name noch nicht im System vergeben ist und die E-Mail-Adresse noch nicht bei einem anderen Nutzer verwendet wird. Implementieren Sie die Methode.

Hinweise: Die Methode liefert 0 zurück, wenn der neue Nutzer registriert wurde, -1, wenn der Benutzername bereits vergeben ist und -2, wenn die E-Mail-Adresse bereits einem anderen Nutzer zugeordnet wurde.

(5 BE)

2 Entwicklung einer Datenbank für die Social-Media-Plattform
Ein Entity-Relationship-Modell (ERM) der Datenbank finden Sie in Material 5.

2.1 Das ERM soll in das relationale Modell überführt werden.

2.1.1 Geben Sie vier Regeln zur Transformation des ERM in das relationale Modell an.

(4 BE)

2.1.2 Überführen Sie das gegebene ERM (Material 5) in das relationale Modell unter Einhaltung der 3. Normalform.

Hinweis: Alle Relationen sind in der Schreibweise `Relation(`<u>`PK`</u>`, Attribut, …, FK#)` anzugeben.

(5 BE)

2.1.3 Referentielle Integrität sichert die Integrität der Daten einer relationalen Datenbank.
Erklären Sie die referentielle Integrität einer Datenbank im Allgemeinen und am Beispiel Ihres relationalen Modells aus Aufgabe 2.1.2.

(4 BE)

2.2 Die Daten der Datenbank sollen ergänzt und ausgewertet werden.

2.2.1 Für die Nutzerin "Anna" sollen alle Titel ihrer Beiträge mit den IDs der enthaltenen Bilder und den Dateinamen der Bilder angezeigt werden.
Formulieren Sie eine entsprechende SQL-Anweisung.

(3 BE)

2.2.2 Der neue Nutzer "Clark", mit dem Passwort "ert4$alPhawe" und der E-Mail-Adresse "clark@family.com" wird der Datenbank heute hinzugefügt. Er erstellt seinen ersten Beitrag (`beitragid` 42) und nutzt dafür das Bild mit der ID 3. Geben Sie die entsprechenden SQL-Anweisungen an.

Hinweis: Die Funktion `NOW()` liefert den Zeitstempel. Fehlende Daten sind sinnvoll zu ergänzen.

(4 BE)

2.2.3 Implementieren Sie eine SQL-Anweisung für die Ermittlung der Titel aller Beiträge mit den Namen der Autoren und der Anzahl der Likes, die im Januar 2023 mehr als 10 Likes bekommen haben, absteigend sortiert nach der Anzahl ihrer Likes.

(5 BE)

2.2.4 Entwickeln Sie eine SQL-Anweisung für die Ausgabe der IDs und Dateinamen aller Bilder, die nicht in Beiträgen verwendet werden.

(3 BE)

2.2.5 Im Folgenden ist ein Ausschnitt einer gewünschten Ergebnistabelle gegeben. In der Tabelle sind die Namen aller Nutzer aufgeführt sowie die Titel ihrer Beiträge, sofern die Nutzer Beiträge erstellt haben.

Benutzername	**Beitragstitel**	**Erstellungsdatum**
Anna	Annas Lieblingsessen	01.01.2023 12:00
Anna	Gerne ein E-Bike!	01.01.2023 14:00
MaxMüller2	happy-end	03.01.2023 01:00
Benedict		
Amelie		
Clark	Best of 2022	20.01.2023 10:00
MaxMüller3		
Don Eric	Porsche 911	23.01.2023 02:00
Martin		

Nachfolgend ist der Entwurf für eine SQL-Anweisung zur Erstellung der gezeigten Ergebnistabelle zu sehen:

```
SELECT benutzerName AS Benutzername, titel AS Beitragstitel,
       erstelltAm AS Erstellungsdatum
FROM   Nutzer N, Beitrag B;
```

Erörtern Sie die Eignung der SQL-Anweisung zur Erstellung der gezeigten Ergebnistabelle. Formulieren Sie eine verbesserte bzw. ergänzte SQL-Anweisung.

(4 BE)

2.3 Die Datenbank soll weitere Anforderungen umsetzen.
Den Betreibern der Social-Media-Plattform ist es gelungen Werbekundinnen und Werbekunden (im Folgenden Werbekunde genannt) zu akquirieren. Diese Werbekunden platzieren Werbespots auf der Plattform und zahlen dafür. Folgende Vorgaben gelten:

– Jeder Werbekunde hat einen Namen und eine Adresse.
– Jeder Werbekunde schließt einen Vertrag über die Platzierung eines Werbespots ab.
– Jeder Vertrag hat eine eindeutige Vertragsnummer, ein Vertragsdatum, eine Laufzeit mit Beginn und Ende sowie den Preis, der pro Platzierung zu zahlen ist, und die Anzahl der Einblendungen des Werbespots pro Tag.
– Jeder Werbespot hat eine Bezeichnung und eine eindeutige ID. Er kann entweder ein Bild oder ein Video sein.
– Jedes Bild hat ein Bildformat und eine Auflösung, die durch die Höhe und die Breite in Pixeln angegeben wird.
– Jedes Video hat ein Videoformat und eine Länge.
– Jeder Werbespot kann Gegenstand mehrerer Verträge sein.

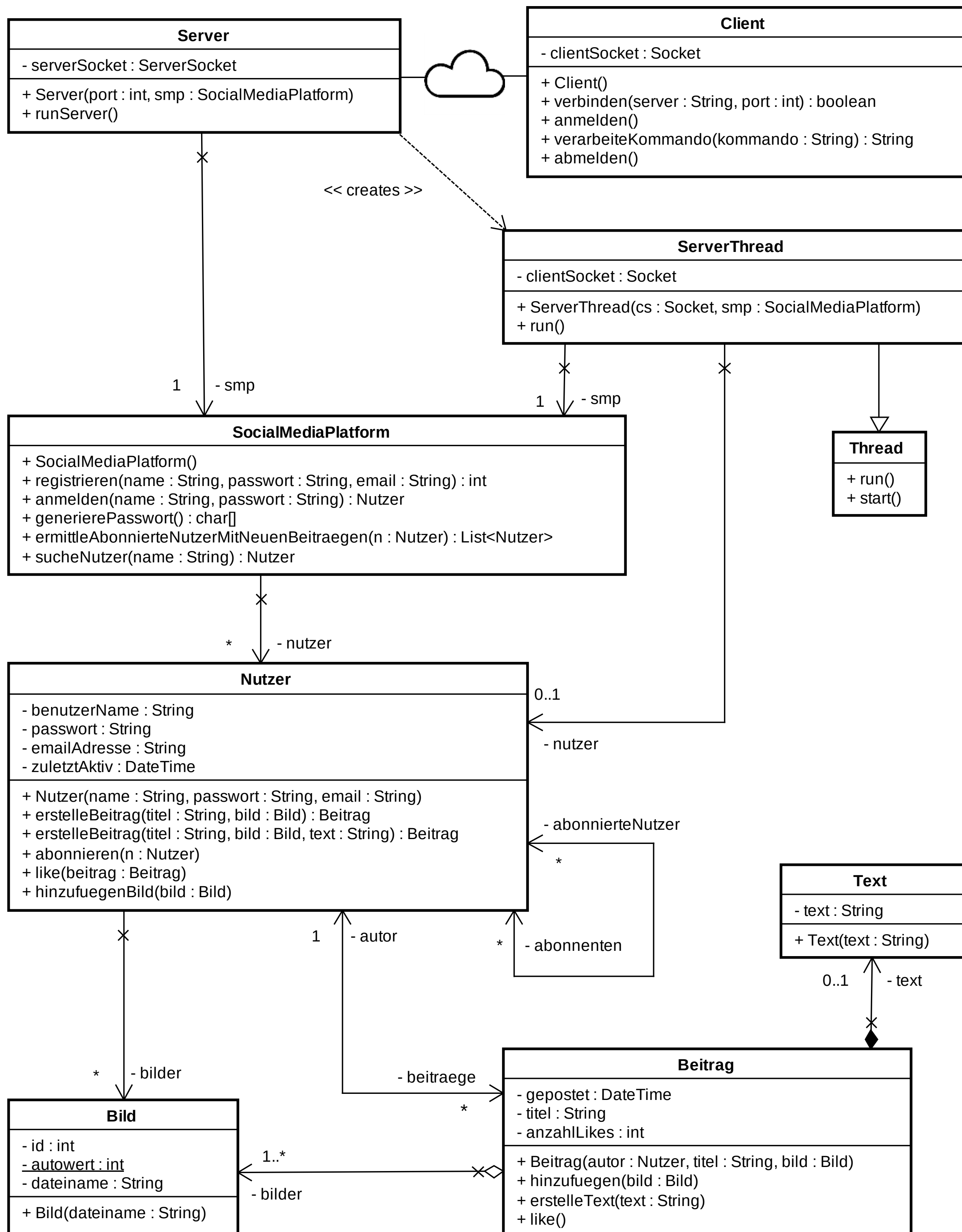
Entwickeln und zeichnen Sie ein neues ER-Diagramm, das die genannten Anforderungen umsetzt.

Hinweise: Jeder Entitätstyp soll einen eindeutigen Schlüssel haben. Das Diagramm ist mit Entitätstypen, Attributen und Beziehungen mit Kardinalitäten in der [min, max]- Notation darzustellen.

(8 BE)

Material 1

UML-Klassendiagramm



Hinweis: Auf alle Attribute kann mittels get-Methoden zugegriffen werden.

Material 2

Klassendokumentationen

Klasse `DateTime`

DateTime
+ DateTime() + isEquals(dt : DateTime) : boolean + compareTo(dt : DateTime) : int + isBefore(dt : DateTime) : boolean + isAfter(dt : DateTime) : boolean + plusMinutes(minutes : long) : DateTime + toString() : String

`DateTime()`
erzeugt ein `DateTime`-Objekt mit dem aktuellen Systemdatum und der Systemuhrzeit, im Format dd.mm.yyyy hh:mm.

`isEquals(dt: DateTime): boolean`
liefert `true`, wenn das DateTime-Objekt gleich dem des Parameters `dt` ist.

`compareTo(dt: DateTime): int`
Vergleicht Datum und Uhrzeit des `DateTime`-Objekts mit dem des Parameters `dt`.
Liefert einen `int`-Wert
< 0 wenn das Datum des `DateTime`-Objekts vor dem des Parameters `dt` liegt,
= 0 wenn das Datum des `DateTime` Objekts gleich dem von `dt` ist,
> 0 wenn das Datum des `DateTime`-Objekts nach dem des Parameters `dt` liegt.

`isBefore(date: DateTime): boolean`
liefert `true`, wenn das `DateTime`-Objekt vor dem des Parameters `dt` liegt.

`isAfter(date: DateTime): boolean`
liefert `true`, wenn das `DateTime`-Objekt nach dem des Parameters `dt` liegt.

`plusMinutes(minutes: long): DateTime`
liefert eine Kopie des `DateTime`-Objekts mit der addierten Anzahl von Minuten.

`toString(): String`
liefert eine String-Repräsentanz des Date-Objekts im Format `"dd.mm.yyyy hh:mm"`.

Klasse `List`

List<T>
+ List<T>() + add(obj : T) + add(index : int, obj : T) + contains(obj : T) : boolean + get(index : int) : T + remove(index : int) : T + remove(obj : T) : boolean + size() : int

`List<T>()`
erzeugt eine generische Liste mit Elementen des Typs `T`.

`add(obj: T)`
hängt das Objekt `obj` vom Typ `T` am Ende der Liste an.

`add(index: int, obj: T)`
fügt das Objekt `obj` vom Typ `T` an der Position `index` in die Liste ein.

`contains(obj: T): boolean`
liefert `true`, wenn das Objekt `obj` in der Liste enthalten ist, sonst `false`.

`get(index: int): T`
liefert das Listenelement an der Position `index` zurück bzw. `null`, falls `index` negativ oder größer gleich der Anzahl der momentan enthaltenen Elemente ist.

`remove(index: int): T`
entfernt das Listenelement an der Position `index`. Liefert das entfernte Element zurück bzw. `null`, falls `index` negativ oder größer gleich der Anzahl der momentan enthaltenen Elemente ist.

Material 2 (Fortsetzung)

`remove(obj: T): boolean`
entfernt das Objekt `obj` aus der Liste. Falls `obj` mehrmals in der Liste enthalten ist, wird nur das erste Vorkommen entfernt. Der Rückgabewert ist `true`, falls das Objekt gefunden und entfernt wurde, sonst `false`.

`size(): int`
liefert die Anzahl der Elemente in der Liste zurück.

Klasse `Random`

Random
+ Random() + nextInt() : int + nextInt(n : int) : int

`Random()`
erzeugt ein Objekt der Klasse Random.

`nextInt(): int`
erzeugt eine Zufallszahl aus dem gesamten Bereich der positiven ganzen Zahlen.

`nextInt(n: int): int`
erzeugt eine Zufallszahl aus dem Bereich 0 (einschließlich) bis n (ausschließlich) der ganzen Zahlen.

Klasse ServerSocket

ServerSocket
- localPort : int
+ ServerSocket(localPort : int) + accept() : Socket + close()

`ServerSocket(localPort: int)`
erzeugt einen Server-Socket und bindet ihn an den angegebenen Port.

`accept(): Socket`
wartet darauf, dass ein Client eine Verbindung aufbauen will. Wurde eine Verbindung aufgebaut, liefert die Methode das entsprechende Socket-Objekt.

`close()`
schließt den Server-Socket.

Klasse `Socket`

Socket
- remoteHostIP : String - remotePort : int
+ Socket(hostIP : String, hostPort : int) + connect() : boolean + readLine() : String + write(s : String) + close()

`Socket(remoteHostIP: String, remotePort int)`
erzeugt ein Socket-Objekt.

`connect(): boolean`
stellt eine Verbindung zum RemoteHost her; liefert `true`, wenn eine Verbindung hergestellt werden konnte.

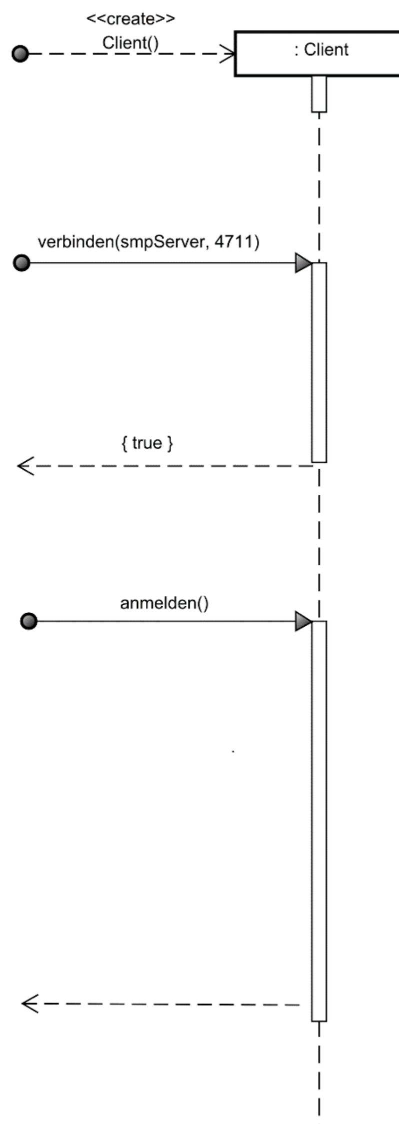
`readLine(): String`
liest eine Zeile vom Socket, bzw. liefert `null`, wenn der Socket nicht geöffnet ist. Eine Zeile wird durch ein Zeilenendezeichen abgeschlossen; das Zeilenendezeichen wird jedoch nicht in den zurückgegebenen String übernommen. Die Methode blockiert, bis eine komplette Zeile eingelesen ist.

`write(s: String)`
schreibt einen String zum Socket; ist die Schnittstelle nicht geöffnet geschieht nichts.

`close()`
löst die Verbindung auf.

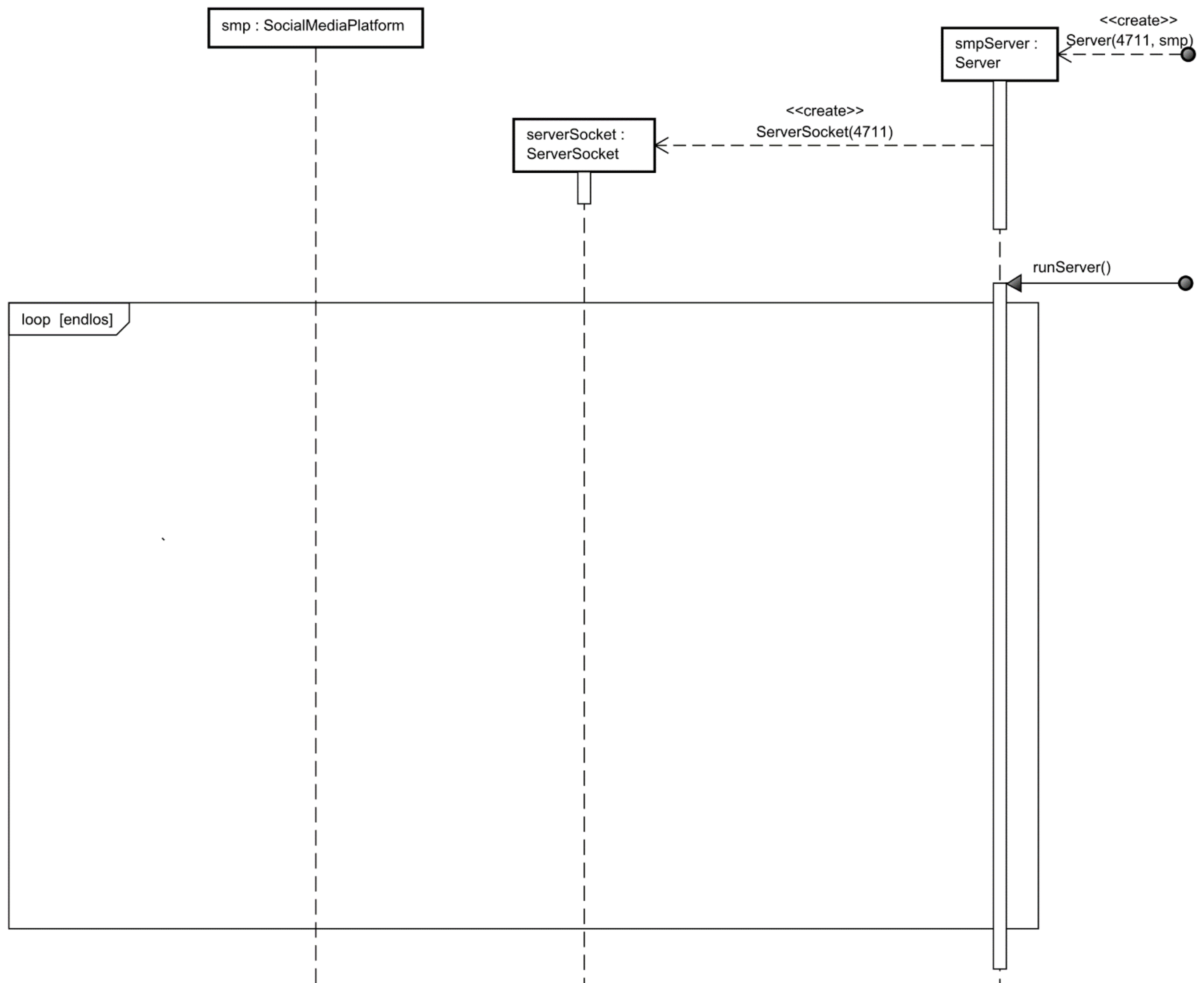
Material 3

UML-Sequenzdiagramm (linker Teil)



Material 3 (Fortsetzung)

UML-Sequenzdiagramm (rechter Teil)



Material 4

Ausschnitt aus einer ASCII-Tabelle

Dez	Zeichen	Dez	Zeichen	Dez	Zeichen	Dez	Zeichen	Dez	Zeichen
33	!	51	3	69	E	87	W	105	i
34	"	52	4	70	F	88	X	106	j
35	#	53	5	71	G	89	Y	107	k
36	$	54	6	72	H	90	Z	108	l
37	%	55	7	73	I	91	[	109	m
38	&	56	8	74	J	92	\	110	n
39	'	57	9	75	K	93	]	111	o
40	(	58	:	76	L	94	^	112	p
41	)	59	;	77	M	95	_	113	q
42	*	60	<	78	N	96	`	114	r
43	+	61	=	79	O	97	a	115	s
44	,	62	>	80	P	98	b	116	t
45	-	63	?	81	Q	99	c	117	u
46	.	64	@	82	R	100	d	118	v
47	/	65	A	83	S	101	e	119	w
48	0	66	B	84	T	102	f	120	x
49	1	67	C	85	U	103	g	121	y
50	2	68	D	86	V	104	h	122	z

Material 5

Entity-Relationship-Modell

